OpenGL: The DirectX Perspective

Course Objectives

- **Quickly orient** DirectX programmers with what core OpenGL has to offer
- **Leverage knowledge** of modern graphics hardware and APIs
- **Update perspectives** for those who still think in terms of OpenGL 1 or 2
- OpenGL programmers will also get a glimpse of the DirectX perspective

Assumed Knowledge

- Modern graphics hardware
  - D3D9 / OpenGL 2 at a minimum
  - Will discuss D3D11 / OpenGL 4.3 features as well
- A modern graphics API
  - Graphics resource management concepts
  - Shader language concepts
- Intermediate real-time rendering knowledge
  - Will not motivate why data from one pass or shader stage would be used in another

Outline

- OpenGL overview
- Data and Operation
- Sychronization
- Shaders

Comparison Slides

OpenGL	D3D
OpenGL 4.2	D3D11

Cross Platform and "Open"

- ARB and Khronos
- Specifications are public and free
  - http://www.opengl.org/registry
  - http://www.opengl.org/documentation/glsl
- Conformance Tests (past/future)
- Extensions
  - Vendor-specific
  - Multi-vendor
  - ARB

OpenGL History

- GL (SGI)
- OpenGL 1: Fixed Function
- OpenGL 2: Programmable
- OpenGL 3: DX10, Cleanup
- OpenGL 4: DX11

- OpenGL ES 1
- OpenGL ES 2
- OpenGL ES 3

- WebGL

Today: OpenGL 4.3

Contexts and Profiles

- Profiles: capability (almost like a device)
- Contexts: Like DX context

- Core profile
- Compatibility profile
- Debug context

Core Contexts

- Introduced with OpenGL 3.2
- Core contexts *remove* legacy functionality
  - Substantial simplification
- Core contexts restrict OpenGL to features actually supported by hardware
- The driver performs less error checking
  - Am I in the middle of a begin-end sequence?
  - Am I compiling a display list?
  - Do I need to validate my fixed-function shader?
- You can mix legacy and core contexts
  - There is a cost associated with switching, so switch rarely (once per frame, perhaps)

Not In Core

- Immediate Mode
- Fixed-Function
- GL Display Lists (not quite the same as DX)
- N-Gons
- Selection + Feedback, accumulation buffers, stipples, …

- Similar to OpenGL ES 2

Debug Context and Output

- Separate debug context for deliberate cost
- Basic use is a callback (no more glError)

```
void callback( enum source, enum type, uint id, enum
        severity, sizei length, const char *message,
        void *userParam );
```

- Or log per context
- Filtering and labelling
- Synchronous or Asynchronous

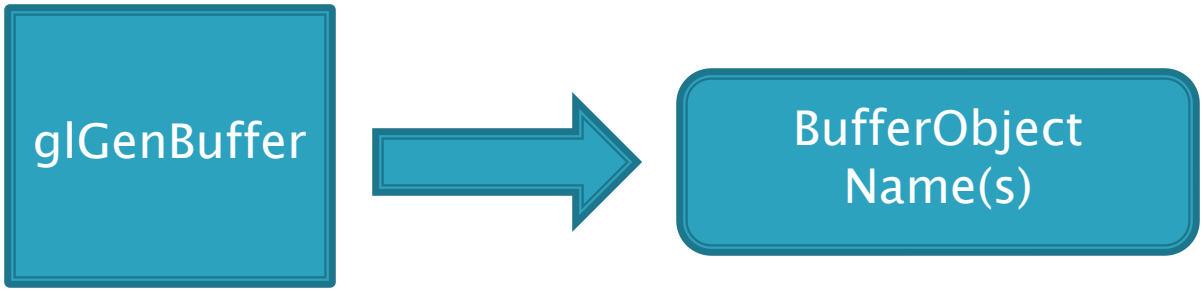
Development Advice

- Develop using core profile with debugging
  - Try different vendor debug contexts
- Benchmark and choose
  - Perpetuate the chicken-and-egg

Resource management and performant OpenGL

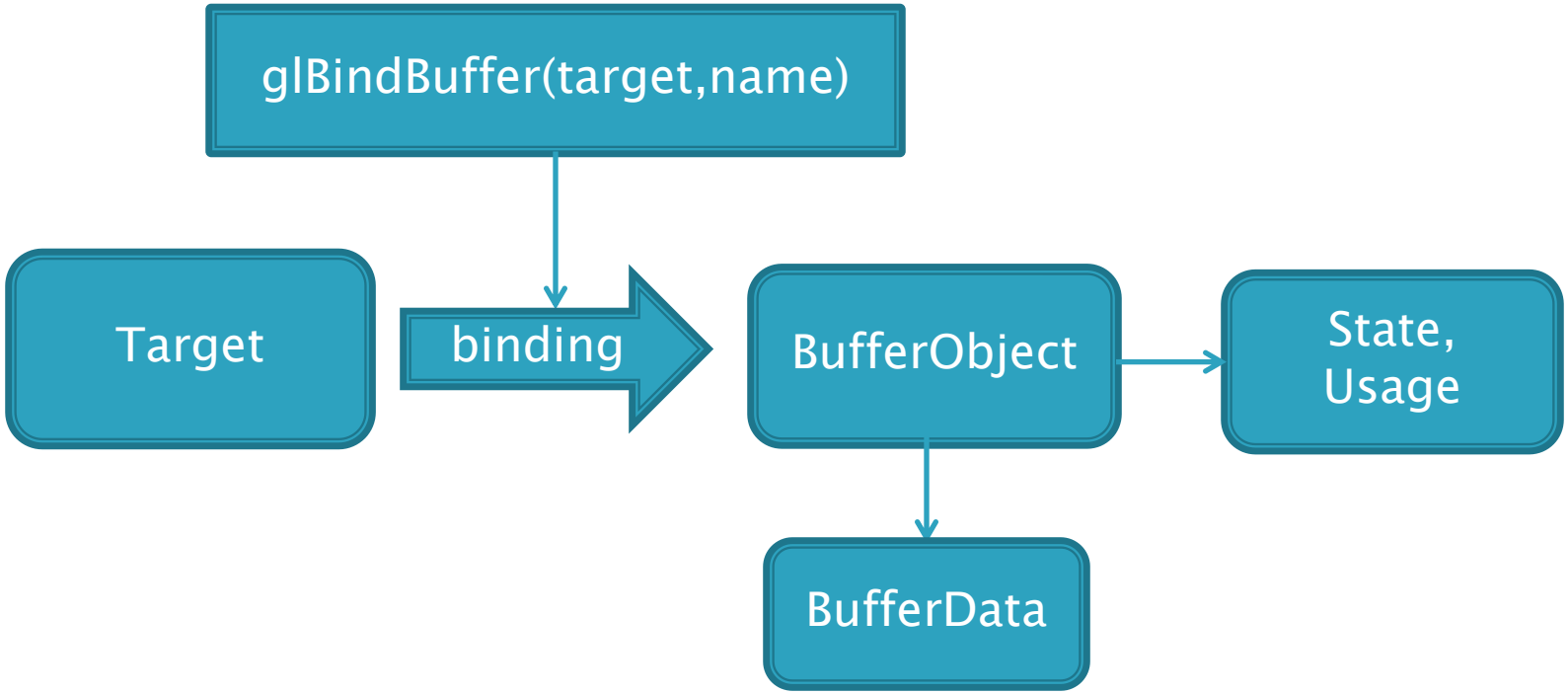
»

Buffers




```
pDevice->CreateBuffer( , , &pBuffer)
```

Buffer Binding and Creation

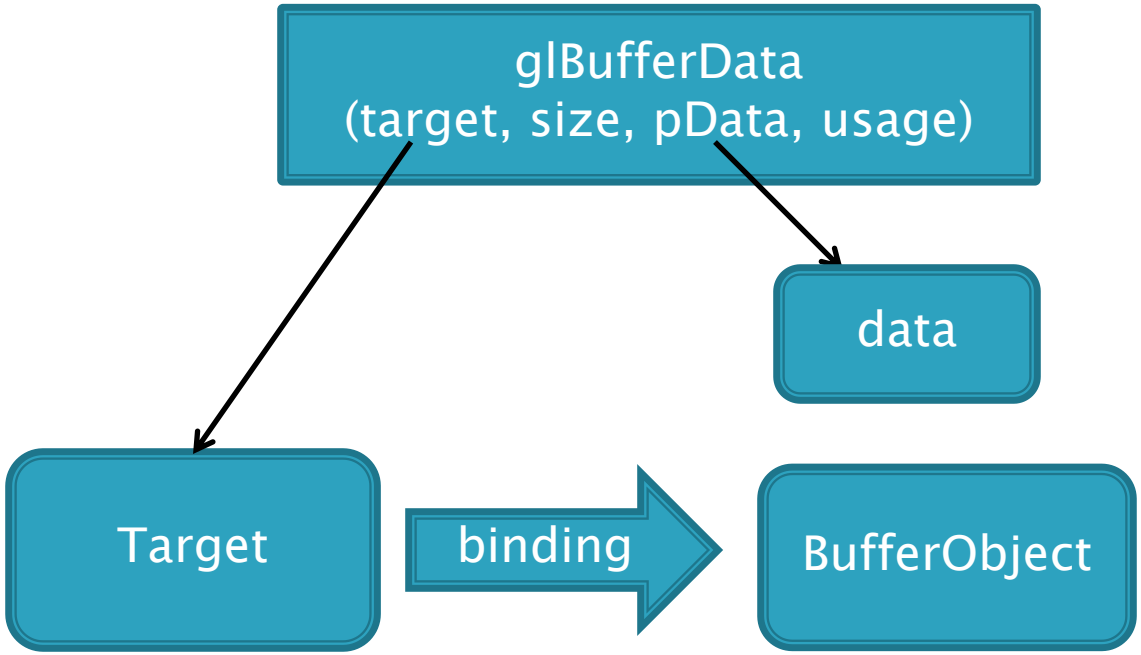


desc.BindFlags = <Target>
pDevice->CreateBuffer(desc,…)

Buffer Targets

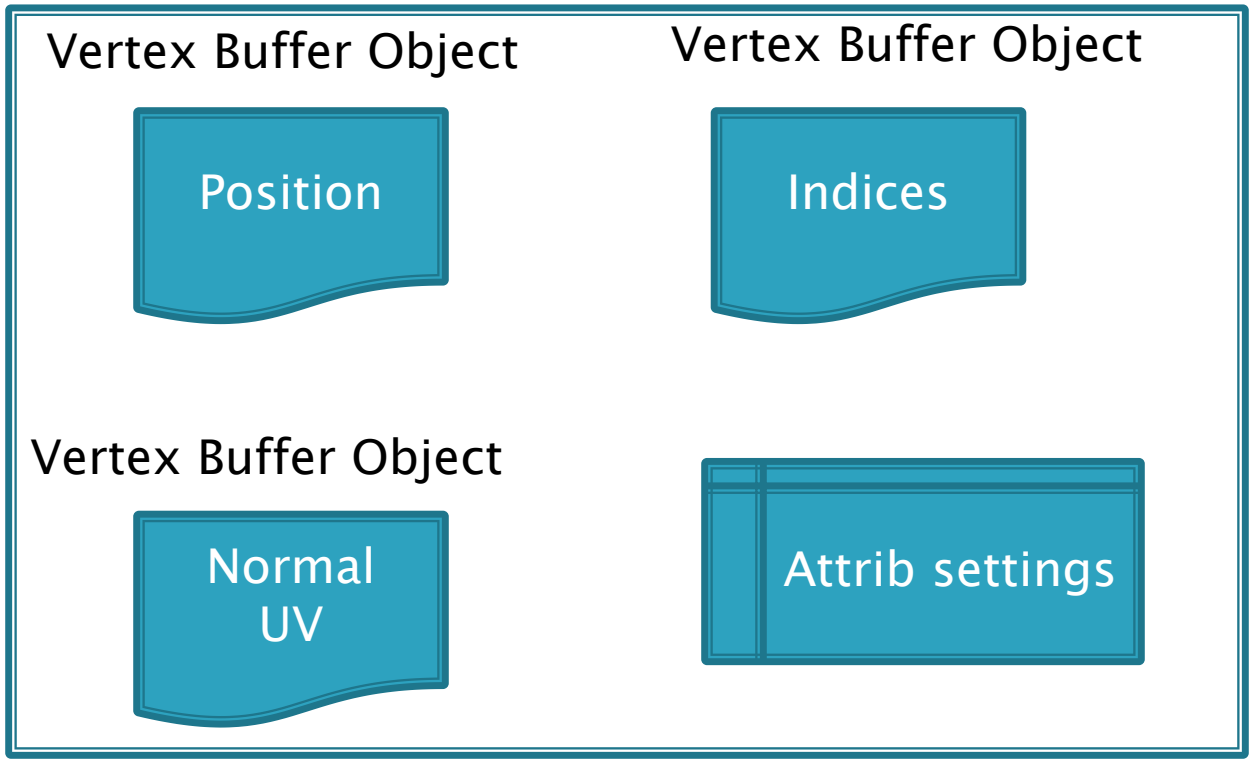
GL Name	Typical Purpose	DX Equivalent
ARRAY	Vertices, generic	VERTEX, SHADER_RESOURCE
ATOMIC_COUNTER	Global counter var	UAV_FLAG_COUNTER
COPY_READ, _WRITE	Copying (optional)	
DRAW_INDIRECT	indirect draw	DRAWINDIRECT
ELEMENT_ARRAY	Indices	INDEX
PIXEL_PACK, _UNPACK	GPU <-> CPU	
TEXTURE	other as Texture	
TRANSFORM_FEEDBACK		Stream out
UNIFORM		CONSTANT

Setting Data (Simplest Option)




```
desc.Usage = <Usage>
desc.CPUAccessFlags = <RWUsage>
pDevice->CreateBuffer(desc,pData,)
```

Vertex Array Objects



D3D: Input Assembler (sort of)

Textures (Immutable)

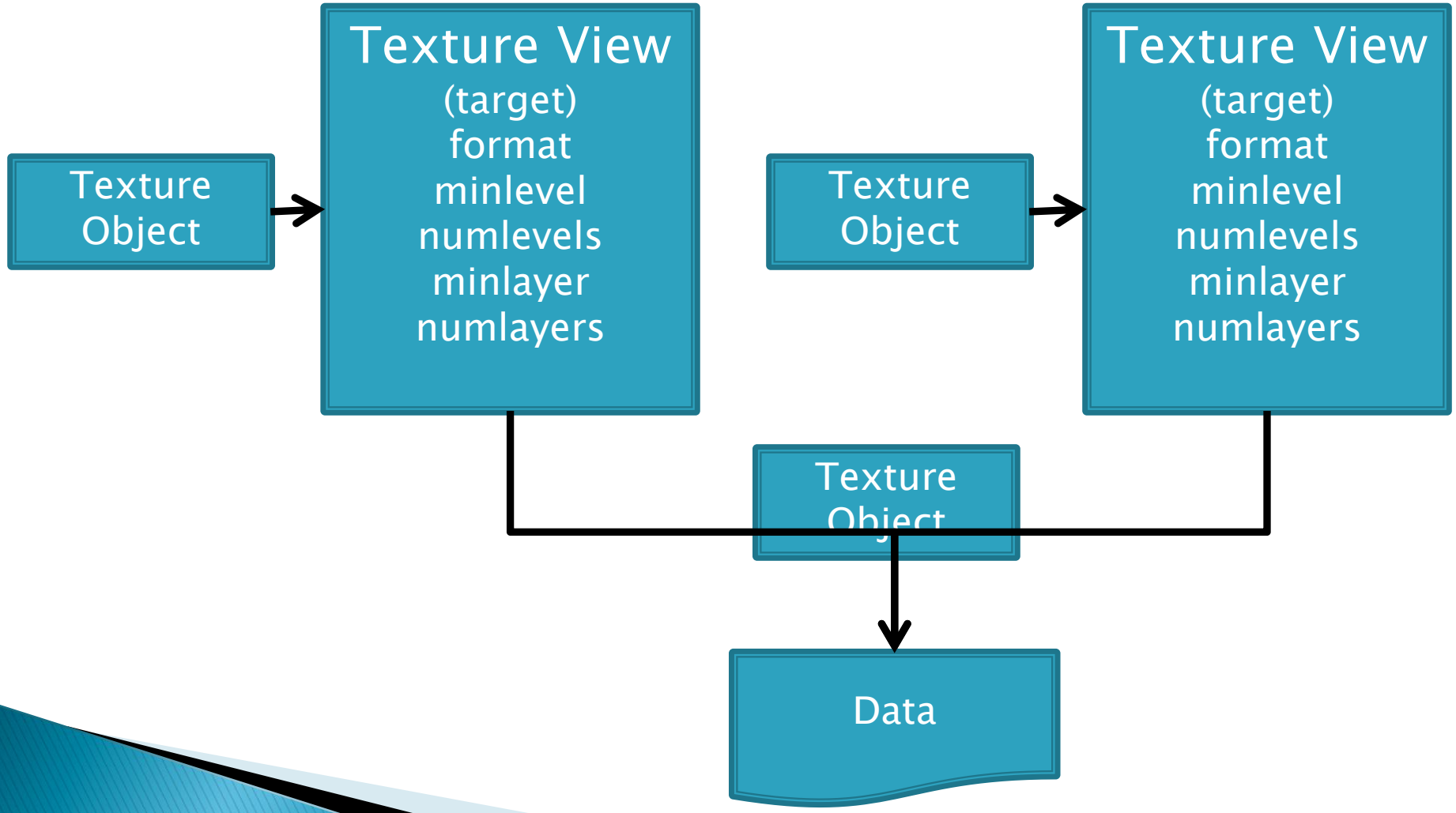
```
glGenTextures();
glBindTexture();
glTexStorage3D(GL_TEXTURE_2D_ARRAY,
    level,   internalformat,
    width, height, depth);
glTexSubImage3D(GL_TEXTURE_2D_ARRAY,
    0,0,0, width, height, depth,
    format, type, pData);
```

```
CreateTexture2D( desc, srcDataLayout, pData);
```

Texture Data Alternatives

- glCopyTexImage
- glTexSubImage
- glCopyTexSubImage

OpenGL Texture Views



OpenGL Sampler State

- Set on texture object (legacy)

```
glTexParameteri(…,GL_MIN_FILTER,…)
```

OR

- Use Sampler object

```
glGenSampler(); glBindSampler();
glSamplerParamteri(…)
```

  - Overrides glTexPrameter when bound
  - Re-usable

Images

- Texture as an indexable array of data
- It's a bind, not an object.
- Not sampled
- Reads and scattered writes (like UAV)
- Atomic operations
- Like tex.Load() in DX

Scattered Writes

- Image load store
- Any pipeline stage
- Like standard buffer
- Some sync control between passes
  - MemoryBarrier(bitfield)
- Between invocations
  - coherent
  - memoryBarrier()

OpenGL 4.2

- UAV
- PS or CS only
- Like RT attachment
- Always synced between Passes
- Sync primitives across group (default)
- globallycoherent for sync beyond group (CS)

D3D11

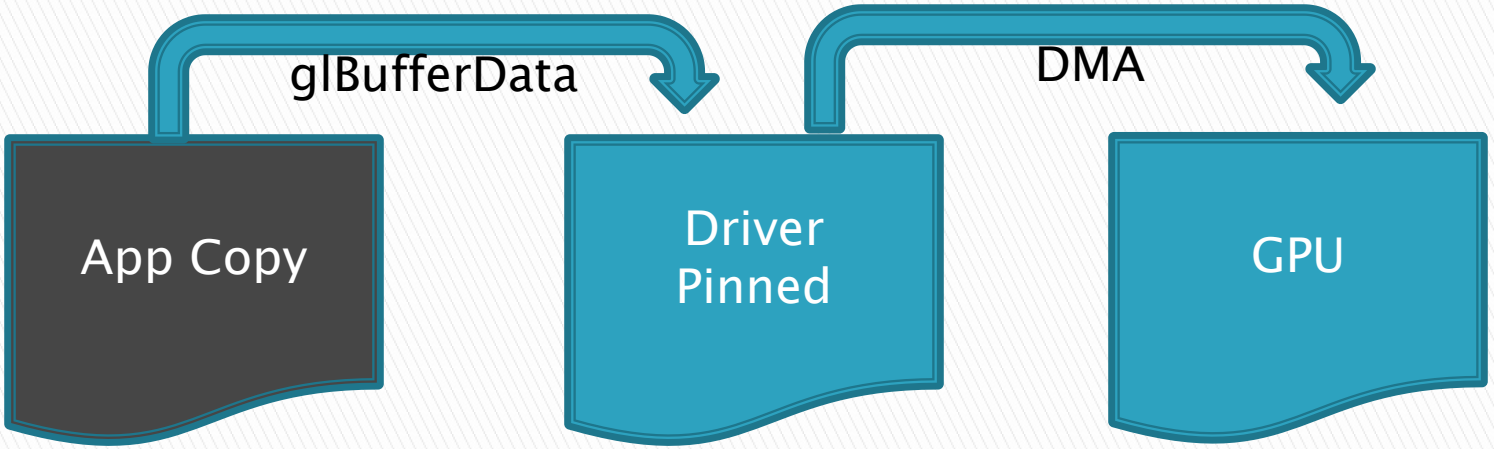
Others

- BufferTexture
  - Buffer Object as texture
- Frame Buffer Object
  - N images, one depth, one stencil
  - Image choices
    - Render Buffer Object
    - Texture (2D slice)
  - OMSetRenderTargets

Draw Calls

OpenGL	D3D
glDrawArrays	Draw
glDrawArraysInstanced	DrawInstanced(…,0)
glDrawArraysInstancedBaseInstance	DrawInstanced
glDrawArraysIndirect	DrawInstancedIndirect
glMultiDrawArrays	for(int i=0; i<n; ++i) Draw(count[i], start[i]);
glMultiDrawArraysIndirect	for(int i=0; i<n; ++i) DrawInstancedIndirect(…)
glDrawElements	DrawIndexed
…And so forth	

Data Transfer: Synchronous Copy



UpdateSubResource

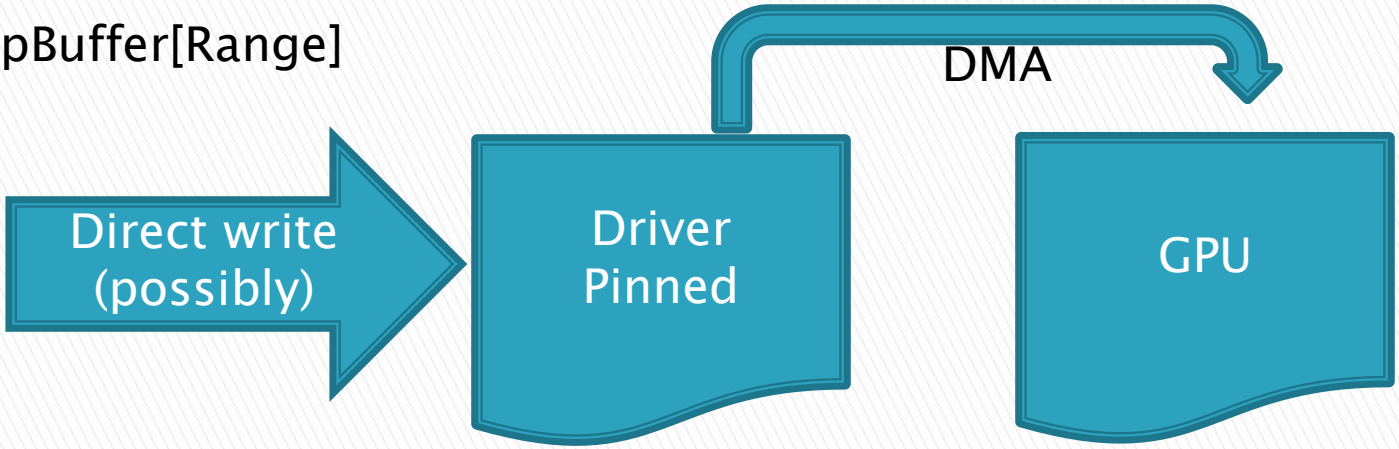
glBufferData

D3D

OpenGL

Data Transfer: Mapped Write

glMapBuffer[Range]

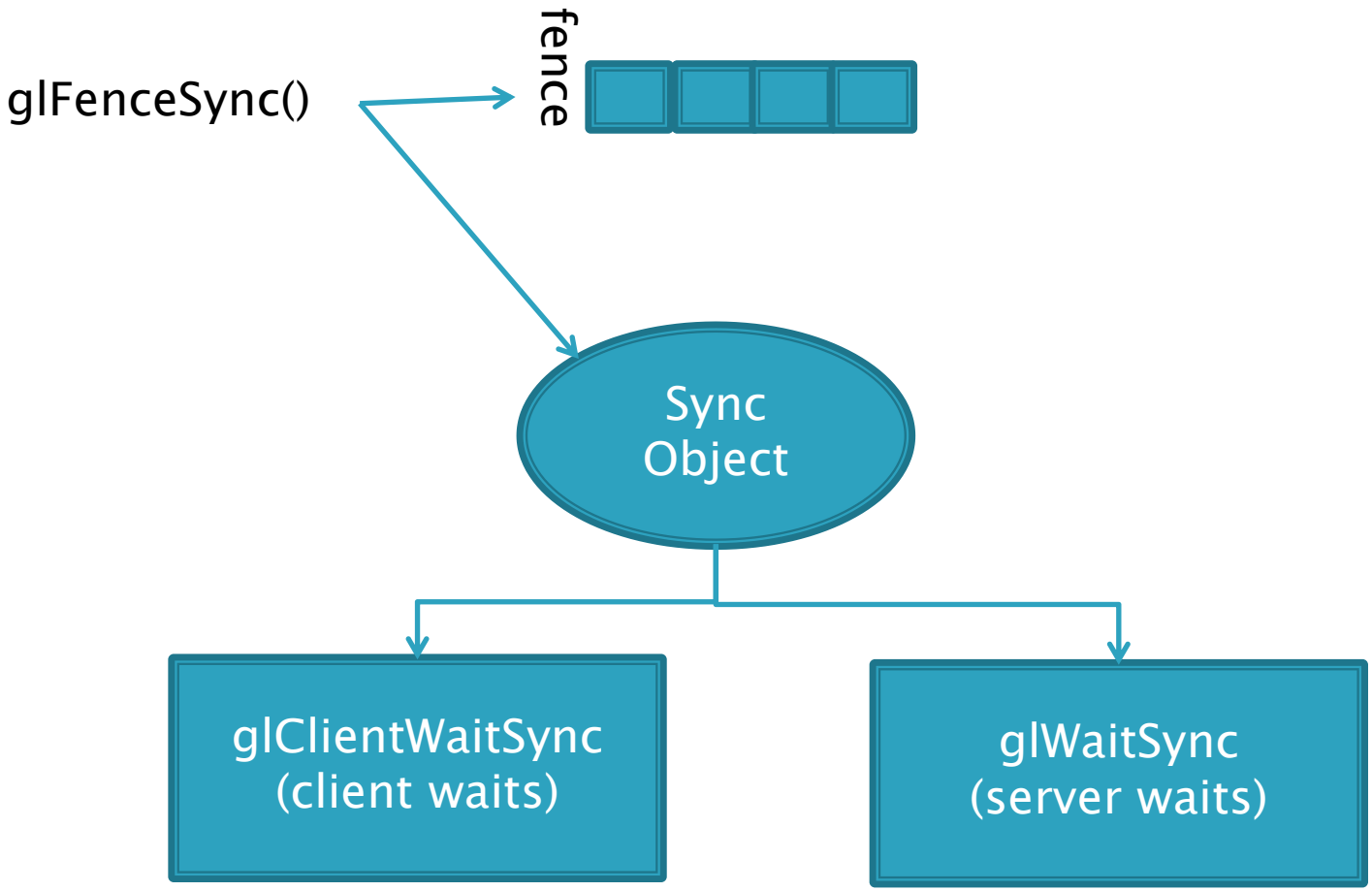


glUnmapMapBuffer[Range]

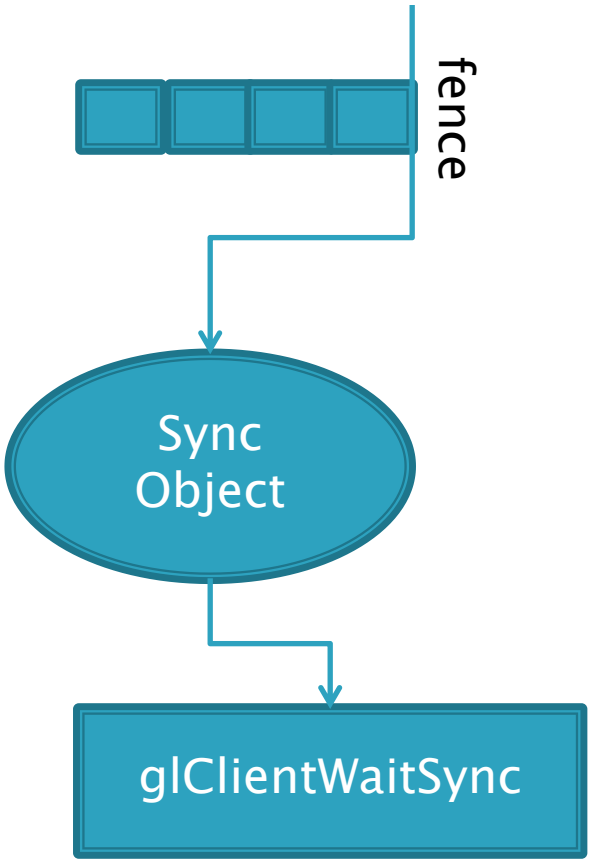
D3D

OpenGL

Command Synchronization



Command Synchronization



Shaders

Shader Types

- Vertex
- Tessellation Control
- Tessellation Evaluation
- Geometry
- Fragment

OpenGL 4.0

- Vertex
- Hull
- Domain
- Geometry
- Pixel

D3D11

Shader Management – DX

```
D3DCompile(source,..,vs_5_0,..,&pByteCode)
CreatVertexShader(pByteCode);
VSSetShader(pShader,0,0);
```

Shader Management – OpenGL

```
glCreateShader(type);
glShaderSource(…);
glCompileShader();

uint program = glCreateProgram();
glAttachShader(program, shader);
glLinkProgram(program);
glUseProgram(program);
```

Program Pipeline

- Combines stages from mulitple programs.

Program Binaries

```
glCreateShader(type);
glShaderSource(…);
glCompileShader();

uint program = glCreateProgram();
glAttachShader(program, shader);
glLinkProgram(program);
glGetProgramBinary(program,…,format,pBinar
yOut);
```

Basic GLSL Vertex Shader

```
#version 430
in vec3 Position;
in vec2 UV;
out VsOutput
{
	vec3 vPositionWS;
	vec2 vUV;
} vs_output;
uniform mat4x4 mVP;
void main(void)
{
	gl_Position = mVP * vec4(Position, 1.0);
	vs_output.vPositionWS = Position;
	vs_output.vUV = UV;
}
```

Basic GLSL Pixel Shader

```
#version 430
in vec3 Position;
in vec2 UV;
out VsOutput
{
	vec3 vPositionWS;
	vec2 vUV;
} vs_output;
uniform mat4x4 mVP;
void main(void)
{
	gl_Position = mVP * vec4(Position, 1.0);
	vs_output.vPositionWS = Position;
	vs_output.vUV = UV;
}
```

Basic GLSL Pixel Shader

```
#version 430

in FsInput
{
	vec3 vPositionWS:
	vec2 vUV;
} fs_input;
uniform sampler2D sDiffuse;
out vec4 color_out;

void main(void)
{
	color_out = texture2D( sDiffuse, fs_input.vUV );
}
```

Basic Geometry Shader

```
#version 430
layout (triangles) in;
layout (triangle_strip, max_vertices = 3) out;

void main(void)
{
     for(int i=0; I < gl_in.length(); i++)
     {
          gl_Position = gl_in[i].gl_Position;
          EmitVertex();
     }
     EndPrimitive();
}
```

Data Types

- int, uint, float, double
- vec3
- mat4x3
- struct
- Opaque types
  - sampler
  - image
  - atomic_uint (counter)

GLSL

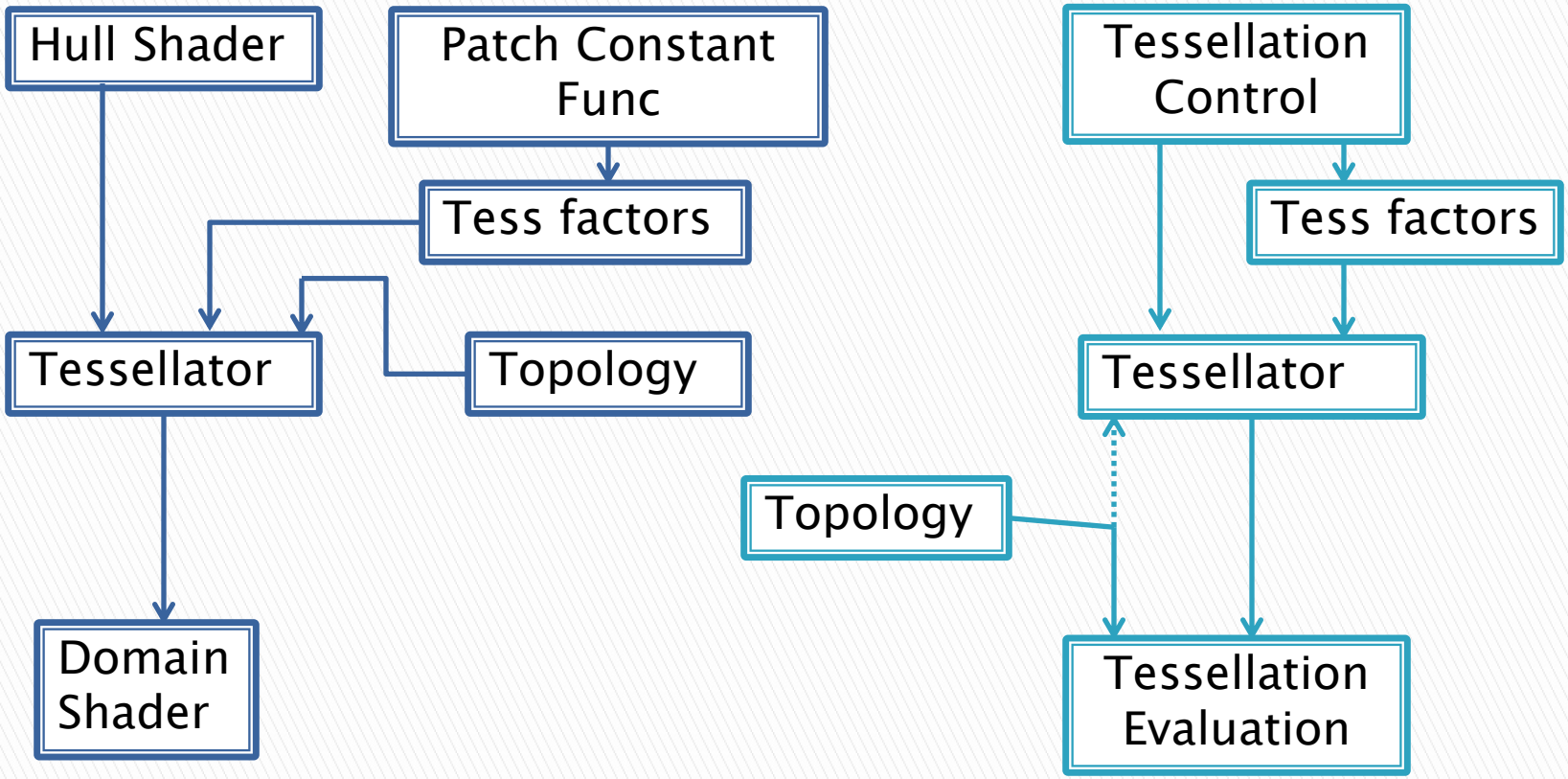
- int, uint, float, double
- float3
- float3x4
- struct
- Objects
  - texture (many variants)
  - buffer (many variants)
  - (buffer method)

HLSL

Layout

- Like HLSL […] in may cases
  - tessellation parameters
  - compute dimensions
- Packing and format
  - layout(shared, column_major)
  - layout(rgba8)
- Bind slots / locations (like registers in HLSL)
  - layout(binding=0)
  - layout(location=0)
- Geometry Shader uses
- Others
  - layout(origin_upper_left) in gl_FragCoord
  - layout(depth_less) // hint for depth mods

Tessellation



D3D11

OpenGL 4.0

Tessellation

```
// Hull Shader
[outputcontrolpoints(4)]
[patchconstantfunc("ConstantsHS")]
[maxtessfactor(16.0)]
[domain("quad")]
[partitioning("integer")]
[outputtopology("triangle_cw")]

HS_OUTPUT HullShader(…)

// Domain Shader
DS_OUTPUT DomainShader(…)
```

```
// Tessellation Control
layout (vertices = 4) out;
void TCS(void)
{
    if (gl_InvocationID == 0)
    {
        gl_TessLevelInner[0] = 2.0;
        …
```

```
// Tessellation Evaluation
layout (quads, cw, equal_spacing) in
void TES(void)
{
    …
```

D3D11

OpenGL 4.0

Tessellation Control

```
out patch float tessFactor;

void main(void)
{
   if (gl_InvocationID == 0)
   {
      gl_TessLevelInner[0] = 2.0;
      …
      tessFactor = 2.0;
   }
   barrier();

   DoSomeWork(tessFactor, gl_InvocationID);
```

Tessellation rate can be set by any instance

Values can be communicated across threads

Compute Shaders

```
glDispatchCompute(nGroupsX,nGroupsY,nGroupsZ)

        Dispatch(nGroupsX,nGroupsY,nGroupsZ)


glDispatchComputeIndirect(offset)

        DispatchIndirect(pResource,offset)
```

OpenGL 4.3

D3D11

DirectCompute

```
RWStructuredBuffer<int> linearOutput;
groupshared int var;

[numthreads(64, 1, 1)]
void ContrivedSample(
    uint3 globalIdx : SV_DispatchThreadID,
    uint3 localIdx : SV_GroupThreadID,
    uint3 groupIdx : SV_GroupID )
{
    if(localIdx.x == 0)
        var = 1;

    GroupMemoryBarrier();

    linearOutput[globalIdx.x] = var;
}
```

OpenGL Compute

```
#version 430
layout(r32i) image1D linearOutput;
shared int var;

Layout(local_size_x = 64, local_size_y = 1, local_size_z = 1)
void main()
{
    const uvec3 localIdx = gl_LocalInvocationID;
    const uvec3 globalIdx = gl_GlobalInvocationID;
    const uvec3 groupIdx = gl_WorkGroupID;
    if(localId.x == 0)
        var = 1;

    Barrier();

    imageStore(linearOutput, globalIdx.x, var);
}
```

Updated Slides

http://developer.amd.com/Resources/documentation/presentations